

Ejercicio 1:

1 a)	TEMP_STATUS \rightarrow 0xFF0	Juego el código queda:	sensor:	CMP [0xFF0], MAX_TEMP
	ATM_STATUS \rightarrow 0xFF1			JG alarma
	WIND_SPEED \rightarrow 0xFF2			CMP [0xFF1], MAX_ATM
				JG alarma
				CMP [0xFF2], MAX_WIND_SPEED
				JG alarma
				JMP sensor
			alarma:	CALL sonarAlarma
				JMP sensor

b) En el caso de que nunca suene la alarma (los valores máximos nunca se alcanzan) NS que la lectura de cada registro de E/S se realice una vez por cada vez que se saltó a sensor. siendo que se saltó a sensor 1 vez cada 7 instrucciones, y todas ellas también se ejecutaron, su frecuencia es de $\frac{1}{76} \text{ Hz}$ y luego el muestreo se realiza con una frecuencia de $\frac{1}{76} \text{ Hz}$.

c) En caso de que al alcanzar la temperatura máxima se produzca una interrupción, el programa quedaría:

sensor:	CMP [0xFF1], MAX_ATM	rutina Int:	CALL sonarAlarma	De esta forma, la rutina de atención a la interrupción <u>suena la alarma</u> (pues se alcanza la temperatura máxima) y vuelve al punto de ejecución.
	JG alarma		IRET	
	CMP [0xFF2], MAX_WIND_SPEED			
	JG alarma			
	JMP sensor			
alarma:	CALL sonarAlarma			
	JMP sensor			

haciendo ahora que el registro de temperatura está ligado a una interrupción, no hace falta hacer su muestreo, por lo que al muestreo de los otros dos se realizará 1 vez cada 5 instrucciones (se reduce la iteración del ciclo suponiendo que los valores máximos no se alcanzan), quedando su frecuencia como $\frac{1}{56} \text{ Hz}$.

Ejercicio 2:

2 a) En caso de que el programa sea interrumpido durante la ejecución de la instrucción ADD R0, R1, primero se terminará de ejecutar la instrucción, lo cual hace que V=1 (se produce overflow al sumar R0=0x0000 con R1=0xFFFF) y luego el estado de los flags en ese punto (PSW) se guardará en el stack para ser restaurado luego de ejecutarse la rutina de atención a la interrupción. Después a ello se realizará todo el proceso para llamar a dicha rutina, donde al ejecutarse la instrucción ADD R0, R1 (con R0=0x0000 y R1=0x0000) los valores de todos los flags serán 0. Sin embargo, como la última instrucción ejecutada de la rutina es RET y no IRET, siendo que tanto el valor del PC y de PSW están en la pila, no se restaurará el PC, con lo cual el estado de los flags en dicho punto será reemplazado por el proveniente de la rutina de atención a la interrupción. Es decir que el flag V pasa de 1 a 0 y que se no se ejecuta la rutina hubo Overflow (lo cual era lo esperado por el salto condicional), ejecuta no hubo Overflow.

b) Para que la ejecución de la rutina de atención a la interrupción sea transparente, habrá que cambiar la instrucción RET por IRET, lo cual hace que no sólo se restablezca el punto de ejecución, sino también el estado de los flags del programa previos a ser interrumpidos. Con esta modificación la ejecución de la instrucción ADD R0, R1 será la esperada, incluso si es interrumpida.

Ejercicio 3:

```

3) while (true) {
    Fetch()           // fetch
    Decode()          // decodificación
    Execute()         // ejecución
    if I==1 AND INTR==1 {
        PUSH PSW      // [SP]=PSW, SP=SP-1
        PUSH PC       // [SP]=PC, SP=SP-1
        CLI           // I=0
        PC=[0x0000]   // llamado a la rutina
        INTA()         // se le indica al dispositivo
                        // que atenderá un pedido
    }
}

```

Ejercicio 4:

```

4) leída Consecutiva: CMP R0, 0x0000 // verifica si se debe seguir leyendo al
                        JE Fin         // dispositivo
leído: MOV R2, [0xFFFF] // verifica si el dispositivo está ocupado
AND R2, 0x0000
JNE leída Consecutiva
MOV R2, [0xFFFF] // escribe el dato a la dirección de memoria
AND R2, 0x00FF // máscara de R2
MOV [R1], R2
ADD R1, 0x0001 // R1 pasa a la dirección siguiente
SUB R0, 0x0001 // decrece R0
JMP leída Consecutiva
Fin: RET // la rutina termina

```

Ejercicio 5:

5) Generar el programa teniendo en cuenta que si el registro DISPLAY-DATA muestra 0xFFFF, al pulsarse a presionar el botón este mostrará 0x0000. De esta manera, el programa se escribe como:

```

main:      MOV [0xFFFF], 0x0000 //inicializa el display
presionado: CMP [0xFF0], 0xFFFF //verifica que el botón se presione
            JNE presionado
soltado:   CMP [0xFF0], 0x0000 //verifica que el botón se suelta
            JNE soltado
            ADD [0xFFFF], 0x0001 //aumenta el valor del display
            JMP presionado //vuelve a ver el estado del botón

```

Ejercicio 6:

- 6 a) Conectamos los pines de la forma: $IR0 \leftarrow$ Presión del aire en acción; $IR1 \leftarrow$ Ollena; $IR2 \leftarrow$ Comodidad; $IR3 \leftarrow$ GPS (el resto desconectados). Esto se debió a la posición de cada uno de los monitores y entres del PIC.
- b) Inicialmente, $IMR = 00001111$ lo cual hace que el PIC pueda atender a todos los interruptores de los monitores.
- c) Vector de interrupciones: $[0xFF00 \ 0xA000 \ 0xFE00 \ 0x01FF \ 0x0000 \ 0x0000 \ 0x0000 \ 0x0000]$ (notar que las últimas 4 posiciones corresponden a las interrupciones no conectadas).
- d) $IRR = 00001010$

```

e) RAI_MONITOR_ALTURA:
    ; Guarda la máscara
    PUSH AX
    IN AX, IMR
    PUSH AX
    ; Setear máscara inhibiendo interrupciones
    ; de menor prioridad
    MOV AX, 0x0001
    OUT IMR, AX
    ; habilitar interrupciones
    SETI
    ; Guarda el valor del registro SI
    PUSH SI
    ; Obtener la altura nueva
    IN AX, 43h
    LEA SI, MONITOR_ALTURA
    MOV [SI], AX
    ; Fin de la rutina
    POP SI
    POP AX
    OUT IMR, AX
    POP AX
    IRET

```


F) Es necesario llamar a la instrucción especial IRET debido que si bien RET restaura el valor del PC antes de producirse la interrupción, es necesario además restaurar el valor del registro de los flags (PSW) ya que estos pueden ser alterados durante la ejecución de dicha rutina. De ambas operaciones se encarga la instrucción IRET, teniendo en cuenta que el valor de dicho registro fue guardado en la pila previo al llamado.

Ejercicio 7:

<pre> 7) a) if (IFH == 1 OR IFL == 1) { PUSH PSW PUSH PC CLEAR HIF CLEAR LIF if (IFH == 1) { PC = [0x0000] INTA() } else { SET HIF PC = [0x0001] INTA() } } </pre>	<p>b) Para completar la arquitectura de dicho procesador sería necesario agregar las instrucciones correspondientes al <u>reset de los flags</u> que habilitan las interrupciones de alta y baja prioridad (HIF y LIF). Esto hace que al atender la rutina de atención de baja prioridad, el programa pueda ser interrumpido por el llamado de la de alta prioridad, y a su vez en la de alta prioridad esto no puede ser interrumpido por uno de baja prioridad.</p>
--	---

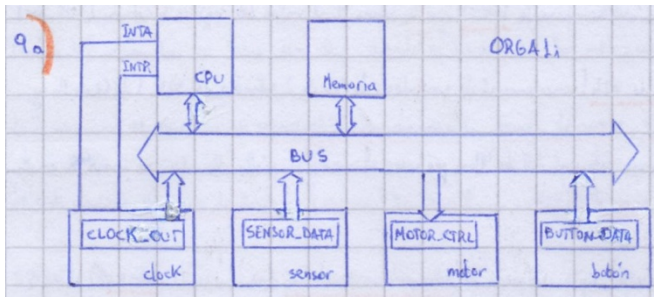
Ejercicio 8:

B) Los pedidos deben ordenarse de la forma: **[Reloj del sistema - teclado - Puerto serial - Disco Rígido - Disco Flexible - Impresora]**

En principio, el reloj es quien se encarga de sincronizar los eventos y operaciones, por lo que se debe ser interrumpido por otro proceso. Luego, el teclado puede ser usado para indicar al procesador la sucesión de operaciones a realizar para controlar en caso de fallas. Seguido a esto está el puerto serial, al cual se pueden conectar una amplia variedad de dispositivos con un tiempo de acceso rápido, al igual que el teclado, pero de menor frecuencia de atención que este.

Por otro lado se tiene el disco rígido, cuyo tiempo de acceso es lento, más aún que el de los dos siguientes, pero de mayor frecuencia en cuanto a su uso. Más allá de que el disco flexible haya caído en desuso, si se lo utiliza este es más prioritario que la impresora.

Ejercicio 9:



b) RAI_CLOCK: ADD R0, 0x000A00
 CMP R0, 0x002D00
 JL habilitada
 deshabilitada: MOV R1, 0x000F00
 CMP R0, 0x005A00
 JL fin-ret
 habilitada: MOV R1, 0x000A00
 fin-ret: IRET

c) while(true){
 sube_barrera = false; // OR boton == 0x000A
 while (NOT sube_barrera){
 if (estado == 0x000A AND boton == 0x0001){
 sube_barrera = true;
 }
 }
 motor = 0x0001; // Sube la barrera
 auto_baja_barrera = false
 while (NOT auto_baja_barrera){
 if (sensor == 0x0001){
 auto_baja_barrera = true;
 }
 }
 sale_auto = false
 while (NOT sale_auto){
 if (sensor == 0x0001){
 sale_auto = true;
 }
 }
 motor = 0x0000; // Baja la barrera
}

```

main:      MOV R0, 0x0000
           MOV R1, 0x000A
fuera-de-hora: MOV MOTOR_CTRL, 0x0000
fuera-de-hora: CMP R1, 0x000A
           JNE fuera-de-hora
fuera-de-hora: CMP BUTTON_DATA, 0x0001
           JNE fuera-de-hora
fuera-de-hora: MOV MOTOR_CTRL, 0x0001
b.presionado: CMP SENSOR_DATA, 0x0001
           JNE b.presionado
auto.pasando: CMP SENSOR_DATA, 0x0000
           JNE auto.pasando
           MOV MOTOR_CTRL, 0x0000
           JMP fuera-de-hora

```

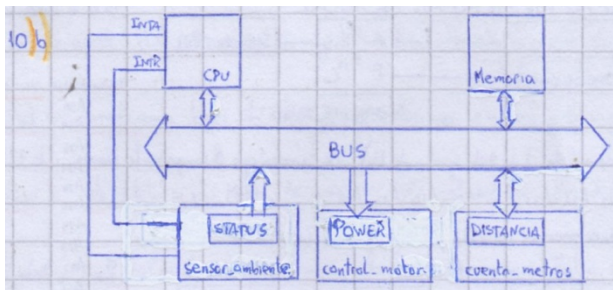
* R0 ← Contador del clock
 * R1 ← Barrera habilitada en 0x000A y sino en 0x000F

Si se procesa $750.000.000$ ns/s se sabe que la frecuencia es de $f = 750 \text{ MHz}$ (frecuencia con la que se ejecuta una instrucción). Luego, una instrucción tarda en ejecutarse $t = \frac{1}{f} = \frac{1}{750} \text{ ns}$. Sabiendo que la rutina de atención a la interrupción (PAI - CLOCK) posee como máximo un ciclo de 9 instrucciones, esta tarda en ejecutarse:

$$T = 9t = \frac{9}{750} \text{ ns} = 0,012 \text{ ns} = 12 \text{ ns}$$

Ejercicio 10:

10) a) [0xFF0] ← SENSOR_AMBIENTE (E)
 [0xFF1] ← CONTROL_MOTOR (S)
 [0xFF2] ← CUENTA_METROS (E/S)




```

c) *rutina_control()
    Km_recorridos = 0; estado_ambiente = 0x0000; control_motor = 0xF000
    while (true)
        if (estado_ambiente == 0x0002) { control_motor = 0xF000 }
        elif (estado_ambiente == 0x0001) { control_motor = 0x30F0 }
        else { control_motor = 0x7FFF }
        if (distancia > 1000)
            Km_recorridos++
            distancia = 0xFFFF
        }
    }
}

*rai()
    if (sensor_ambiente == 0xFFFF) { estado_ambiente = 0x0002 }
    elif (sensor_ambiente == 0x0001) { estado_ambiente = 0x0001 }
    else { estado_ambiente = 0x0000 }
}

```

; Km recorridos y el estado del
 ; ambiente comienzan en nulos
 ; Se controla el motor del
 ; tren en base al estado
 ; de ambiente
 ; Si se recorrió un km, la
 ; Km recorridos incrementa
 ; en 1

; Según el valor del sensor del
 ; ambiente, se determina el
 ; valor del estado del ambiente

```

d) *main:
    MOV DISTANCIA, 0xFFFF
    MOV R0, 0x0000
    MOV [0xFFFF], 0xF000
    MOV R1, 0x0002
    sensor_freno: CMP R1, 0x0001
                  JE sensor_prec
                  JL sensor_cruc
                  MOV [0xFFFF], 0xF000
    sensor_prec:  INP dist 0x0001
    sensor_cruc: JNE sensor_cruc
    sensor_prec: MOV [0xFFFF], 0x30F0
                  JMP dist
    sensor_cruc: MOV [0xFFFF], 0x7FFF
    dist:        CMP DISTANCIA, 0x03EB
                  JL sensor_freno
                  MOV DISTANCIA, 0xFFFF

```

	ADD R0, 0x0001	
	JMP sensor_freno	
* RAI_ambiente:	CMP [0xFF0], 0x0000	
	JG precucion	
	JE cruceo	
	MOV R1, 0x0002	
precucion:	JMP fin_ret	
	MOV R1, 0x0001	
	JMP fin_ret	
cruceo:	MOV R1, 0x0000	
fin_ret	IRET	

R0 ← Kilómetros recorridos
R1 ← Estado del ambiente
DISTANCIA ← [0xFF2] (cuenta metros)